1.3 Programming Assignment Questions

1. Source: wikipedia.org

Raw moments M_{ij} for Image $I(x,y)$ (grayscale)

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

$$\text{Area} = M_{00} = \sum_x \sum_y x^0 y^0 I(x,y) = \sum_x \sum_y I(x,y)$$

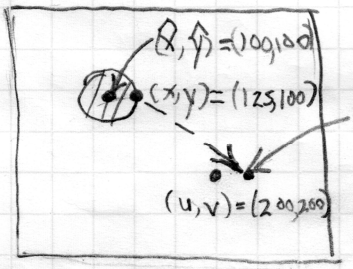
which is just the sum of all the pixels

$$\text{centroid} = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right) = (\hat{x}, \hat{y})$$

2. Assuming that the object is a constant pixel value so visual center is the same as the centroid $\{\hat{x}, \hat{y}\}$

$$dx = u - \hat{x}$$
$$dy = v - \hat{y}$$
$$x_T = x + dx$$
$$y_T = y + dy$$

Ex Image I



$(x_T, y_T) = (225, 200)$

$\dim(I) = 256 \times 256$

$$dx = 200 - 100 = 100$$
$$dy = 200 - 100 = 100$$
$$x_T = 125 + 100 = 225$$
$$y_T = 100 + 100 = 200$$

1, 3

3.

Step 1

$X_{s_i} = \alpha \cdot X$

$y_{s_i} = \alpha \cdot y$

compute initial scale points for object and centroid $(\hat{x}, \hat{y})$

Step 2

compute centroid for new scaled object, all points (x_{s_i}, y_{s_i}), equal to $(\hat{x}_{s_i}, \hat{x}_{s_i})$

Step 3

compute delta for how center has moved, and add to all scaled points

$dx = \hat{x} - \hat{x}_{s_i}$

$dy = \hat{y} - \hat{y}_{s_i}$

$x_s = x_{s_i} + dx$

$y_s = y_{s_i} + dy$

Note: original centroid minus intermediate scaled centroid

yield new scaled object (x_s, y_s) still centered at $(\hat{x}, \hat{y})$. It works because this is what I did in my program Assignment 3.

4. Translate Image (see contents lab3.zip)

Wrote program:

TranslateScaleImage/TranslateScaleImage/main.c

Before image: Data/Part1_result_before.png

After image: Data/Part1_result_after.png

1.3

5. Two pictures of sets, different scale and background

Data/Part2_result_close.png
Data/Part2_result_far.png

second image is interesting because face_cascade_detectMultiScale() function though part of door was face.

6 Trajectory of object on top of final image.

Data\Part3_result.png

7. Shape on top of faces.

Data\Assignment3_Result.png

1.4 Written Questions

1, (a) Sorting the number by threshold gives

6/10 —	1.4658	1
7/10 —	1.0305	1
8/10 —	0.6718	1
7/10 —	-0.3682	0
8/10 —	-0.4165	1
7/10 —	-1.2355	0
8/10 —	-1.3566	1
7/10 —	-1.6207	0
6/10 —	-1.776	0
5/10 —	-1.9628	0

The true count of
one's = 5
zero's = 5

The threshold that maximizes percent correctly classified

All of these give 8/10 correct

Any $-1.2355 < T < -0.4165$ or
$-0.3682 < T < 0.6718$ or
$-1.6207 < T < -1.3566$

1.4 1.(b) The threshold T from (a) yields 1 false positive and 1 false negative

(c) Any $-0.3682 < T < 0.6718$ gives a perfect detection of the "1" class

(d) A perfect detection of the "1" class means zero false positives

2. The size of the filter depends on the size of the dots, but assuming it is a single black pixel and using a 3x3 kernel

Filter

1/4	1/2	1/4
1/2	-3	1/2
1/4	1/2	1/4

Black Dot

1	1	1
1	0	1
1	1	1

$\Sigma = 3$ (max)

To normalize, add 3 to the resulting image and then divide by 6.

White Dot

0	0	0
0	1	0
0	0	0

$\Sigma = -3$ (min)

3. I would invert the result of the convolution

4. Either change the size of the kernel - bigger kernel to look for bigger dots. Or rescale the image.

5, Mean Filter replacing pixel with mean of neighborhood

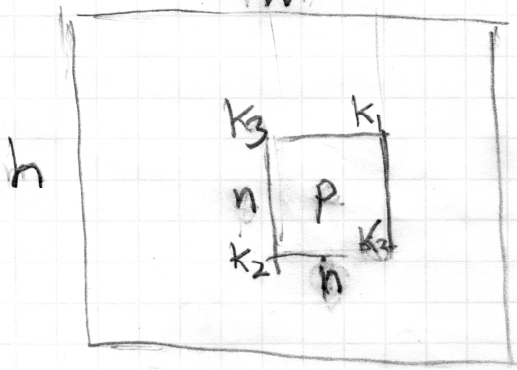
1/8	1/8	1/8
1/8	0	1/8
1/8	1/8	1/8

Including center pixel in mean

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

6 For any $n \times n$ mean filter, put $1/n$ into each cell

7. I would use the integral image to compute the sum of an $n \times n$ region of pixels in the neighborhood of each pixel. Then I would divide by n^2 and replace the pixel in the center of the kernel with the result



For each pixel p in the $h \times w$ image, compute

$$p = \frac{k_4 - (k_1 + k_2) + k_3}{n^2}$$

8. Step1) For an image I the integral image I_i Use I_i to compute $\sum x_i$ for a pixel neighborhood, then square the result divided by n

Step2) Square each pixel in $I \rightarrow I_2$ and then compute an integral image of $I_2 \rightarrow I_{2i}$

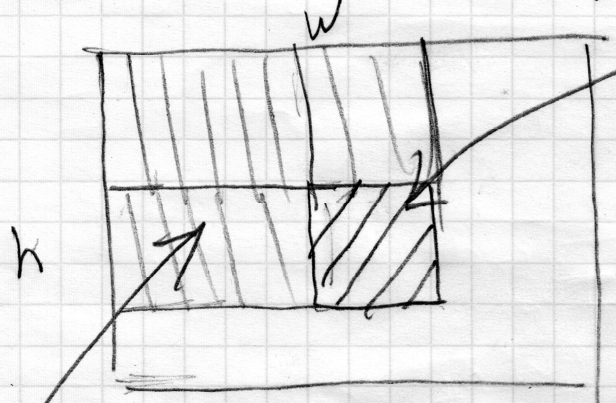
Use I_{2i} to compute $\sum(x)^2_i$ for a pixel neighborhood then divide by n

Step3) Take difference from step1 minus value from step2

9. I would use the discrete kernel K_{LoG} representing the Laplacian of the Gaussian that capitalized on associative property (gaussian ∘ laplacian) ∘ image. Kernel (9x9) Fig3

I would take the sum of that Kernel and then create a new Kernel K_{LoG} the contain the weighted averages based on the original

1.4 9. . . . cells in K_{LoG}.



In this region use weighted averages in K_{LoG} WA as a look up table to compute the weighted avg for particular pixel under kernel times sum of whole area under kernel found by using Integral image calculations.

These other 3 region use std. integral calc. to get sum under kernel.

This would give approx LoG result with assumption the pixels under 9x9 Kernel had small variance. The larger the variance of pixels in original image under kernel window, the less valid the resulting KoG.